

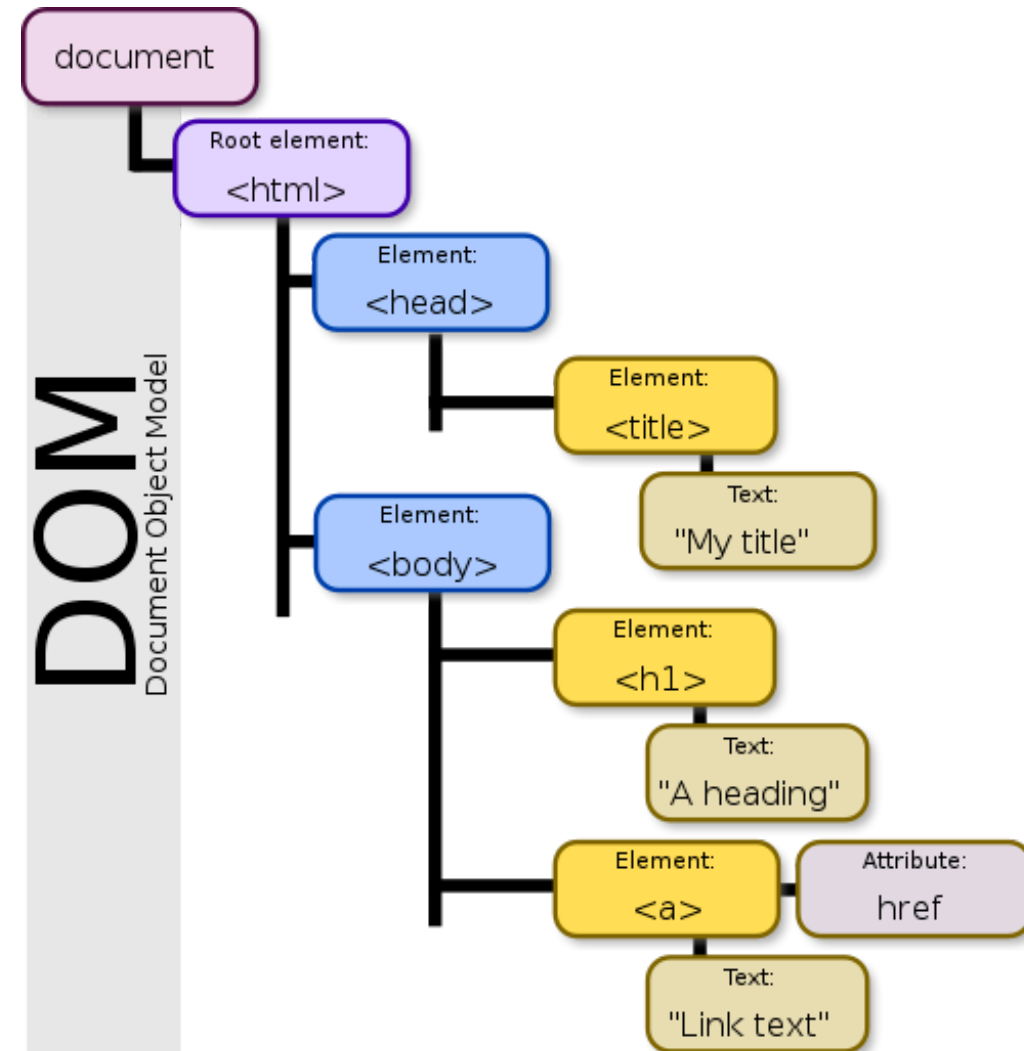
Le DOM pour les nuls IDIA



Version PDF des slides

DOM : les principes

- Document Object Model = représentation du document sous forme d'arbre
 - Documents HTML, XML, SVG, etc.
- **Document** : le document lui même
- **Document.documentElement** : la racine du document (`<html>`)
- L'arbre est composé de (types de base) :
 - **Node** : l'élément de base
 - **Element** : élément / "balise"
 - **Attribute** : attribut d'un élément
 - **NodeList** : un tableau de Noeuds...
- Le texte contenu dans un élément est un simple noeud "texte"



DOM : les principes

- Chaque type de document possède ses propres types spécifiques
 - Ex HTML: [HTMLElement](#), [HTMLImageElement](#), etc.
 - Ex SVG : [SVGElement](#), [SVGCircleElement](#), etc.
- Pour les documents XML on utilise les types de base
- Le DOM permet de parcourir, consulter, modifier un document

DOM : quelques outils

- Récupération d'un élément (`document.querySelector()`) ou plusieurs (`document.querySelectorAll()`) à partir d'un **sélecteur CSS**

```
const conteneur = document.querySelector('#conteneur');  
const sects = document.querySelectorAll('section');
```

DOM : quelques outils

- Création d'éléments avec `document.createElement(nomDeTag)` ou modification de leur contenu texte (`textContent`) ou HTML (`innerHTML`)

```
const para = document.createElement('p');  
  
para.textContent = 'Un petit texte sympa.';  
// ou  
para.innerHTML = '<a href="#">Un lien</a>';
```

DOM : quelques outils

- Manipulation des attributs avec `getAttribute()` ou `setAttribute(attribut, valeur)`
- Si on ne peut pas utiliser de classes ou d'id on utilise `style`

```
para.setAttribute('class', 'highlight');  
para.style.color = 'red';  
const attr = para.getAttribute('class');
```

DOM : quelques outils

- Ajout d'un élément comme fils d'un autre avec `appendChild(élément)`
- Suppression d'un fils avec `removeChild(élément)`
- Suppression directe d'un élément avec `remove()`

```
conteneur.appendChild(para);  
conteneur.removeChild(para);  
// ou  
para.remove();
```

- `Document`, `Node`, `Element`, `HTMLElement` possèdent de nombreuses autres méthodes et propriétés
- Chaque `élément HTML` possède ses propres spécificités

Evenements : principes

- `addEventListener(typeEvenement, callback)` : Enregistre une fonction (callback) qui sera appelée lors de l'événement
 - Préciser le `type d'événement` en 1er paramètre

```
// on récupère un bouton via le DOM
let btn=document.querySelector('#my-button');

// Ajout d'un écouteur d'événement au 'click'
btn.addEventListener('click', () => { // callback sous forme de fonction flèche

    // Petite boîte de message pour l'utilisateur
    alert('You clicked me!');

    // Puis ajout d'un paragraphe à la fin du body
    let pElem = document.createElement('p');
    pElem.textContent = 'This is a newly-added paragraph.';
    document.body.appendChild(pElem);

});
```

Evenements : propagation

- Notion d'[event bubbling](#):
 - L'élément HTML qui gère un événement n'est pas forcément celui qui l'a généré
 - Appel des gestionnaires d'événement en cascade (enfant vers parents)
 - [Event.target](#) = l'élément à l'origine de l'événement
 - [Event.currentTarget](#) = l'élément dont le gestionnaire d'événement a été appelé
 - [Event.stopPropagation\(\)](#) stoppe la propagation d'un événement
- Exemple:

```
let form = document.querySelector('#ex-bubble form');  
  
form.addEventListener('click', (event) => {  
  event.target.style.backgroundColor = 'yellow';  
  alert("target = " + event.target.tagName  
    + ", currentTarget=" + event.currentTarget.tagName);  
  event.target.style.backgroundColor = ''  
});
```

